VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Absolvování individuální odborné praxe

Individual Professional Practice in the Company

2013

Dominik Moravec

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra informatiky

Zadání bakalářské práce

Student: **Dominik Moravec**

Studijní program: B2647 Informační a komunikační technologie

Studijní obor: 2612R025 Informatika a výpočetní technika

Téma: Absolvování individuální odborné praxe
Individual Professional Practice in the Company

Zásady pro vypracování:

1. Student vykoná individuální praxi ve firmě: Tieto Czech s.r.o.
2. Struktura závěrečné zprávy:
a) Popis odborného zaměření firmy, u které student vykonal odbornou praxi a popis pracovního zařazení studenta
b) Seznam úkolů zadaných studentovi v průběhu odborné praxe s vyjádřením jejich časové náročnosti
c) Zvolený postup řešení zadaných úkolů
d) Teoretické a praktické znalosti a dovednosti získané v průběhu studia uplatněné studentem v průběhu odborné praxe
e) Znalosti či dovednosti scházející studentovi v průběhu odborné praxe
f) Dosažené výsledky v průběhu odborné praxe a její celkové zhodnocení

Seznam doporučené odborné literatury:

Podle pokynů konzultanta, který vedl odbornou praxi studenta.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **Doc. Mgr. Jiří Dvorský, Ph.D.**

Konzultant bakalářské práce: Kristian Wiglasz

Datum zadání: 16.11.2012

Datum odevzdání: 07.05.2013

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA * Fakulta elektrotechniky a informatiky * OSTRAVA *

doc. Dr. Ing. Eduard Sojka
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Souhlasím se zveřejněním této bakalářské práce dle požadavků čl. 26, odst. 9 *Studijního a zkušebního řádu pro studium v bakalářských programech VŠB-TU Ostrava.*

V Ostravě 6. května 2013 .....

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 6. května 2013 .....

Rád bych na tomto místě poděkoval firmě Tieto Czech s.r.o, že mi umožnila vykonání Bakalářské praxe, dále všem členům týmu, ve kterém jsem pracoval a v neposlední řadě také mé rodině za psychickou podporu.

Abstrakt

Tato bakalářská práce shrnuje průběh mojí bakalářské praxe ve firmě Tieto Czech s.r.o, která poskytuje komplexní služby v oblasti IT pro soukromý i veřejný sektor. Po dobu mé praxe jsem se podílel na refactoringu aplikace určené pro koncové zákazníky severských energetických firem. V úvodu této práce se vyjadřuji k mému rozhodnutí o absolvování odborné praxe. Dále následuje odborné zařazení firmy, její produkty a popis mého pracovního zařazení. Následuje seznam zadaných úkolů a architektura aplikace, které se všechny zadané úkoly týkaly. V další části popisuji řešení úkolů a také zevrubný popis technologií, které jsem využil. V úplném závěru této práce uvádím, co jsem touto praxí získal, jaké praktické znalosti z mého dosavadního studia jsem využil, ale také jaké znalosti mi scházely.

Klíčová slova: Odborná praxe, Tieto, .NET, Unit testy, MVC, JavaScript

Abstract

This thesis summarizes my stay on Professional Practice in the Tieto Czech s.r.o Company, which providing full life-cycle services for both private and public sectors. During the practice I participate on refactoring application for end users of nordic energy companies. I write about my decision why I choose Professional Practice. Next is description of Company specialization, Company products and my assignment. Thesis continue with list of given tasks to resolve and application architecture explaining. All my tasks involved this application. In the next part of this thesis I explain resolving of the given tasks and technologies that I used. At the end part I writing about experiences that I got during the practise, what skills I used and what skills I missed.

Keywords: Professional practice, Tieto, .NET, Unit testing, MVC, JavaScript

Seznam použitých zkratek a symbolů

AJAX	–	Asynchronous JavaScript and XML
API	–	Application Programming Interface
DOM	–	Document Object Model
FBS	–	Fake Backend System
HTML	–	Hyper Text Markup Language
HTTP	–	Hypertext Transfer protocol
IE	–	Internet Explorer
IT	–	Information Technology
JS	–	JavaScript
JSON	–	JavaScript Object Notation
MS	–	Microsoft
MVC	–	Model–View–Controller
SVG	–	Scalable Vector Graphics
UT	–	Unit Testy
VML	–	Vector Markup Language
XAML	–	Extensible Application Markup Language
XML	–	Extensible Markup Language

Obsah

Seznam obrázků

Seznam výpisů zdrojového kódu

1 Úvod

Když jsem se rozhodoval, zda budu vykonávat odbornou praxi nebo vypracovávat zadané téma, zamyslel jsem se především nad tím, co pro mě bude přínosnější. Jelikož mě baví programování a rád bych hledal své budoucí uplatnění jako programátor, rozhodl jsem se pro absolvování odborné praxe. Mezi hlavní faktory mého výběru patří také skutečnost, že spousta dnešních firem požaduje předchozí praxi a znalosti moderních technologií. Další přínos vyplývá z práce mezi zkušenými programátory, poznání vnitřní struktury firmy a nabytí cenných zkušeností při práci v týmu. V tomto ohledu jsem usoudil, že správnou volbou bude odborná praxe a nikoliv vypracování zadaného tématu, které v mém případě nepřinese tolik užitku. Dalším krokem bylo vyhledání firmy, která bude splňovat tyto ohledy, a proto jsem prošel několika přijímacími pohovory. Nejlepší volbou se mi jevila firma Tieto Czech s.r.o, do které jsem také nastoupil jako software developer a byl zařazen do vývojového týmu.

2 Odborné zaměření firmy a zařazení studenta

Společnost Tieto, založená v roce 1968, se sídlem v Helsinkách a čistými tržbami 1,8 miliard EUR zaměstnává na 17 000 expertů a působí ve více než 20 zemích světa. Do České republiky společnost Tieto vstoupila v roce 2001 a v roce 2004 otevřela své softwarové centrum v Ostravě. S více než 1 900 zaměstnanci je jedním z největších zaměstnavatelů v oblasti IT služeb v České republice a největším v Moravskoslezském kraji.

2.1 Odborné zaměření firmy

Firma Tieto Czech s.r.o nabízí IT řešení v mnoha oblastech. Své služby poskytuje středně velkým a velkým společnostem a k tomuto využívá mnoho moderních technologií různých firem jako IBM, Microsoft, Oracle, SAP atd. Níže se nalézá seznam odvětví, ve kterých firma Tieto působí.

- Energetika
- Finanční služby
- Lesní průmysl
- Telekomunikace
- Výroba
- Maloobchod
- Vládní instituce
- Média
- Zdravotnictví a sociální péče

2.2 Popis mého pracovního zařazení

Po přijetí firmou Tieto, jsem absolvoval Induction days, což je název pro jakýsi typ vstupního školení, které je rozděleno do dvou dnů. Zde jsme získali informace o struktuře firmy Tieto, o používání informačního systému Tieto intra a mnoho dalších informací. Po té jsem se seznámil se svým Line Managerem a byl představen týmu, do kterého jsem byl zařazen na pozici Software Developer. Tento tým se nazývá Online a to z prostého důvodu - vyvíjí a udržuje software pro koncové zákazníky severských energetických firem. Tyto služby se nazývají self-services. Zákazník v tomto systému má možnost prohlížet např. spotřebu energií za určitý časový interval nebo požádat o nový kontrakt (zažádání o novou přípojku) atd. Analogicky by se tento systém dal přirovnat k některému z internetových bankovnictví, kde má zákazník banky možnost jak prohlížet stav svého účtu, tak částečně spravovat svůj účet. Kromě opravných balíčku a programování nové funkcionality se tento tým zabývá i přenesením této aplikace do novějších technologií a zde také začíná

mé uplatnění v tomto týmu. Před mým příchodem tento tým navrhl seznam úkolů, které budou určeny pro studenty na odborné praxi. Některé z těchto úkolů mi byly přiděleny a také budou rozepsány níže. Podle rozsahu byly úkoly vypracovány samostatně či paralelně s jinými studenty. Každý úkol měl svého vedoucího (kmenového zaměstnance), který dohlížel na prováděný úkol, a se kterým bylo možno tento úkol konzultovat.

3 Zadané úkoly

Všechny mé úkoly se týkají webové aplikace, vyvíjené týmem Online, která, jak jsem již popsal výše, je postupně převáděna do novějších technologií.

3.1 Unit testy a Fake backend systém

Díky architektuře, na které je postavená aplikace vyvíjená týmem Online, je možné napsat unit testy dále jen „UT“. Pro tyto účely je využit Fake backend systém dále jen „FBS“, což je jinými slovy databáze stojící mimo reálný systém. Základní ideou tedy bylo vytvořit framework pro automatizaci shormažďování reálných dat do FBS, dále vytváření a naplňování tabulek pro UT a v neposlední řadě vytvořit automatizované UT. Mým úkolem bylo tuto myšlenku realizovat a naučit ostatní členy týmu jak používat FBS a jakým stylem psát nové UT.

3.2 Porovnání grafových knihoven jqPlot a HighCharts

Naše aplikace využívá javaScriptovou knihovnu jqPlot pro vykreslování grafů, která je zdarma pro jakékoliv využití. Z důvodů přenášení aplikace do novějších technologií bylo navrženo využít jinou knihovnu, za účelem zjednodušení kódu a příslušných konfiguračních objektů. Mým úkolem tedy bylo porovnat knihovnu jqPlot s knihovnou HighCharts. Zjistit výhody/nevýhody a nakonec implementovat HighCharts do naší aplikace a pokusit se o nalezení lepší cesty implementace, než je současná.

3.3 Převod prezentační vrstvy do MVC

Pro zlepšení pozdější údržby aplikace bylo navrženo implementovat prezentační vrstvu do Microsoft MVC oproti staré implementaci v Microsoft ASP.NET Web Forms a velké části javaScriptu. Právě kvůli JavaScriptu vznikal „špagetový kód“, který velmi ztěžoval pozdější údržbu prezentační vrstvy. Mým úkolem bylo navrhnout a implementovat prezentační vrstvu do MVC.

4 Architektura aplikace

Na začátek by bylo vhodné zmínit, že se jedná o webovou aplikaci, jejíž serverová část je založená na technologiích Microsoftu a klientská část pak implementována především ve skriptovacím jazyce. Celá aplikace týmu Online je rozložena do více vrstev, což je určitě vhodné pro snadné modifikace či záměny různých vrstev aplikace. Dalším důvodem je také přehlednost celé aplikace, což je důležité z důvodu údržby. Projekt je postaven na architektuře ASP.NET a všechny informace o klientech jsou načítány z globálních backend systémů nazývaných CAB a UMS ve formátu XML. Architektura těchto backend systému mi není známá, jelikož jde o projekt jiných týmů. Tato informace však není pro mě důležitá, jelikož naše aplikace s těmito systémy komunikuje pomocí rozhraní. Logika aplikace je pak vytvořena v jazyce C# a uživatelské rozhraní tvoří z velké části javaScript z důvodu uživatelsky přívětivých efektů. Níže uvádím zobecněný popis jednotlivých vrstev aplikace.

4.1 Datová vrstva

Tato vrstva je postavena na nejnižší úrovni celé aplikace, má za úkol vytvořit připojení k backend systémům CAB a UMS, které navenek komunikují přes svá rozhraní. Obsahuje také sadu transfer objektů, které slouží pro naplnění požadovanými daty z backend sytémů a následnému předání vyšším vrstvám. Transfer objekty jsou výhodné, protože šetří konektivitu mezi aplikací a backend systémem. Idea je taková, že jedním voláním získáme požadované informace v jednom objektu (např. informace o zákazníkovi na základě jeho id), místo nevhodného opakovaného volání pro každý atribut zvlášť. Tyto objekty bývají serializovány pro přenos po síti, což v našem případě obstarává serializace do formátu XML. Poslední důležitou součástí datové vrstvy je Connection manager, který má za úkol vybrat backend systém, se kterým proběhne komunikace. Výhodou tohoto přístupu je možnost přidání nového backend systému bez dalších zásahů do kódu aplikace.

4.2 Business vrstva

Tato vrstva obsahuje aplikační logiku a slouží k získávání a zpracovávání dat vrstvy podřízené, tedy datové vrstvy. Obsahuje také business entity, což jsou typicky objekty popisující entity reálného světa (např. Customer, DeliverySite, Contract atd.). V naší aplikaci jde v praxi o objekty, které obsahují metody vracející na základě parametru transfer objekty datové vrstvy. Protože je business vrstva této aplikace dosti rozmanitá, bylo vhodné vytvořit rozhraní pro prezentační vrstvu. Pro tyto účely posloužil návrhový vzor fasáda (viz 4.2.1). Aby bylo rozhraní fasády co nejpřehlednější a co nejjednodušší na implementaci, bylo třeba vytvořit další komponentní objekty. Tyto objekty obsahují metody pro business logiku. Tyto metody jsou následně využívány ve fasádě.

4.2.1 Návrhový vzor Fasáda

Tento vzor popisuje řešení jak zjednodušit rozhraní podřízené vrstvy v aplikaci. Pokud máme klienta (prezentační vrstvu) a k němu vrstvu podřízenou (business vrstvu), můžeme jednoduše využívat v klientovi jednotlivá rozhraní business objektů. Pokud je, ale business vrstva rozmanitá a obsahuje mnoho tříd, může být používání rozhraní jednotlivých business objektů komplikované, proto je vhodné definovat jednotné rozhraní (fasádu), pomocí kterého budeme komunikovat s business vrstvou. Obecně tedy můžeme říci, že fasáda je jakýsi typ rozhraní, který poskytuje metody vyšší úrovně a odstiňuje tak klienta od složitosti implementace nižší vrstvy.

4.3 Prezentační vrstva

Prezentační vrstva je nejvyšší vrstvou této webové aplikace. Tvoří webové rozhraní pro uživatele sloužící pro komunikaci se systémem. Je navržena především pro zákazníky energetických společností za účelem správy svého uživatelského účtu. Zákazníci těchto společností tak získávají přehled nad svou spotřebou energií za určitý interval, a to formou grafu či tabulky. Dále mají možnost podávat různé žádosti přes webový formulář či jinak spravovat svůj účet. K implementaci prezentační vrstvy bylo na serverové straně využito Microsoft ASP.NET Web Application a na straně klienta skriptovací jazyk JavaScript a jeho knihoven jako např. jQuery, jqPlot atd. Pro komunikaci s business vrstvou využívá její fasádní rozhraní (viz. 4.2 a 4.2.1).

4.4 Knihovna Common

Protože bylo nutno využívat různé komponenty, které by nebylo vhodné zařazovat do vrstev aplikace, byla vytvořena knihovna Common, která je zařazena v Solution a nepředstavuje aplikační vrstvu. Již podle názvu knihovny je zřejmé, že obsahuje různé pomocné třídy pro ostatní vrstvy aplikace. Pro představu zde uvedu pár součástí knihovny. Nachází se zde např. Session Manager, který slouží pro jednotný přístup k Sessions. Dale pak Configuration Manager, který nastaví příslušnou konfiguraci aplikace z XML souboru do session, případně získá konfiguraci podle jména souboru. Tyto konfigurace bývají typické pro konkrétní energetickou firmu, která tento software využívá, což je výhodné, protože není nutné při distribuci software novému zákazníkovi (energetické Společnosti) upravovat zdrojový kód aplikace. Protože je aplikace distribuována do různých zemí, byl vytvořen také Localization Manager, který jednoduše získá informaci o jazyku internetového prohlížeče, ve kterém aplikace běží a na základě těchto informací zvolí vhodný resource soubor pro překlad aplikace do daného jazyka. Knihovna Common také obsahuje framework pro automatizaci unit testů, což byla součást jednoho z mých úkolů.

5 Řešené úkoly

Po příchodu do týmu Online jsme byli seznámeni s architekturou aplikace a také se seznamem úkolů, který je připraven pro studenty na odborné praxi. Zde bych také rád doplnil, že po dokončení každého úkolu byl svolán meeting, na kterém jsem prezentoval své výsledky. Jelikož část našeho týmu se nacházela ve Finsku, bylo třeba tento meeting vést částečně online a prezentovat v angličtině. Meeting obvykle pokračoval volnou diskuzí ohledně tématu, dále se řešili dotazy ostatních členů týmu a návrhy na vylepšení. V poslední fázi se určilo, zda je nutné pokračovat na úkolu či provést navrhnutá vylepšení, a nebo úkol uzavřít jako hotový.

5.1 Unit testy a Fake backend systém

První úkol, který jsem mimochodem vykonával paralelně s jiným studentem v našem týmu, se týkal testování naší aplikace. Díky architektuře aplikace, kterou jsem popsal výše (viz. 4), bylo možné vytvořit sadu Unit testů, což sice může zabrat dost času, ale na druhou stranu získáte automatizovaný nástroj pro kontrolu funkčnosti kódu aplikace.

5.1.1 Co je to Unit test

Jednoduše řečeno jde o kód, který otestuje požadovanou funkčnost jiného kódu (jednotky). Jednotkou je myšlena určitá část kódu aplikace např. funkce, třída atd. Nelze otestovat objekt třídy jako takový, proto zpravidla píšeme testy pro metody tříd, u kterých chceme ověřit jejich funkčnost. Níže uvádím jednoduchý příklad Unit testu.

```
public boolean TestSouctu()
{
        Kalkulacka calc = new Kalkulacka();

        if (calc.secti(5,3) != 8)
        {
                return false;
        }
        else
        {
                return true;
        }
}
```

Výpis 1: Příklad Unit testu v jazyce C#

5.1.2 Motivace

Proč vytvářet Unit testy? Na tuto problematiku lze pohlížet dvěma přístupy:

- Test-first
- Code-first

Test-first - V praxi se využívá tak, že programátor na základě definovaných požadavků napíše sadu testů, které neprocházejí a následně programuje výslednou aplikaci oproti testům. Jakmile programátor dojde do fáze, kdy testy procházejí, zaměří se na optimalizaci kódu a spouští testování znova. Výhodou je, že pokud jsou testy jednou napsány, je velmi jednoduché je spouštět při refaktorizaci a optimalizaci znovu. Pokud se vyskytne chyba, je okamžitě odhalena neúspěšným testem. Nevýhodu pak je, že testy mohou zabrat dost času při samotném vytváření a také mohou obsahovat chyby.

Code-first - Programátor nejdříve vytvoří aplikaci a po té vytváří sadu Unit testů, které podle daných požadavků procházejí. Tímto získá velký nástroj pro případnou optimalizaci či refactoring kódu, který je pokryt testy. Při každé úpravě kódu, může programátor spustit testy a zjistit tak, jestli jeho zásah do kódu neměl negativní vliv na požadovanou funkcionalitu.

V týmu Online byla po mém příchodu aplikace již nějaký čas hotová a nasazená u mnoha zákazníků, proto jediná cesta jak napsat Unit testy byla Code-first. Jak jsem ale již naznačil dříve, celý projekt se předělává do novějších technologií, a proto hlavní motivací pro Unit testy byl refactoring kódu. Každý den se pracuje na nové verzi programu, a proto bylo vhodné vytvořit sadu UnitTestů, která by testovala funkčnost upraveného kódu a zabránila tak nevědomému zanášení chyb do aplikace. Dalším důvodem jsou opravy chyb v kódu či vytváření opravných balíčků. Může se stát, že při opravě jedné chyby se v systému objeví nová chyba. Pokud však máme správně napsané unit testy, je tato chyba okamžitě odhalena.

5.1.3 Návrh Unit testů pro webovou aplikaci

V našem případě bylo nejlepší řešení vytvořit Unit testy na straně serveru. Všechny testované metody musejí mít návratovou hodnotu, aby bylo možné získat nějakou hodnotu pro porovnání testem, který tuto porovná s příslušným záznamem a rozhodne, zda je kód v pořádku či nikoliv. Problém může nastat, pokud kód testované metody pracuje se záznamem z databáze. Může totiž dojít k situaci, kdy testovaná metoda zpracuje záznam z databáze a vrátí nějaký výsledek, ale mezitím dojde v databázi k aktualizaci údaje a při porovnání výsledků test neprojde, což je samozřejmě zapříčiněno vzniklou nekonzistencí dat. Tomuto chybovému stavu je třeba se vyvarovat, a to nejlépe tak, že vytvoříme testovací prostředí s reálnými daty, které ale budou po dobu testu neustále konzistentní.

5.1.4 Fake backend systém

Elegantním řešením, jak předejít nekonzistenci dat při unit testech je využití falešného backend systému se stejným rozhraním jako má pravý backend. Do toho systému jsou poté nahrána testovací data z pravého backendu. Všechna data v pravém backend systému jsou reprezentovaná formátem XML. Proto i náš fake backend systém musel tuto vlastnost zohlednit. Fake backend se skládá z MS SQL databáze, třídy UnitTestDataProvider, která simuluje rozhraní reálného backend systému a všechna data ukládá ve formátu XML do MS databáze a několika dalších pomocných tříd.

5.1.5 Microsoft Unit testing framework

Měli jsme dvě možností, jak Unit testy napsat. Buď vytvořit vlastní framework pro přípravu testů nebo využít hotový framework. Microsoft disponuje poměrně kvalitním frameworkem pro testování aplikací, a proto jsme jej v našem projektu využili. Tento Framework nabízí testování na různých vrstvách aplikace např. na uživatelském rozhraní nebo na business vrstvě serverové strany. Pravě serverová strana nás bude zajímat u naší aplikace, a to konkrétně fasádní vrstva. Microsoft u toho přístupu doporučuje využít falešnou databázi pro konzistentní data (viz 5.1.4). Jeden z důvodů využití tohoto frameworku je možnost načítání scénářů testů z databáze.

5.1.6 Testovací scénáře

Unit test framework nabízí třídu TestContext, která umožní načítání dat z databázových tabulek na základě názvu sloupce, ukázku uvádím níže.

```
string expected = (string)TestContext.DataRow["expectedResult"];
```

Výpis 2: Příklad použití třídy TestContext

Do těchto scénářových tabulek jsou primárně ukládány výsledky metod (a pár dalších parametrů), které budou testovány. Poměr je přesně 1:1 tzn. jedna metoda na fasádní vrstvě odpovídá jedné scénářové tabulce. Každá tato tabulka má stejný název jako testovaná metoda a k tomuto názvu je přidán počet parametrů, aby nedocházelo ke konfliktům při použití přetížených metod. Tento přístup má tu výhodu, že si můžeme předem připravit data, která budou testována. K tomu aby test věděl z jaké tabulky data vybírat, je třeba nastavit každé testované metodě vlastní DataSource, ve kterém je nastavená cesta k databázi a název scénářové tabulky.

5.1.7 Testovací metody

V těchto metodách již provádíme samotné testování. Je tedy třeba získat výsledek testované metody ze scénářové tabulky přes TestContext, jak jsem popisoval výše, a dále zavolat testovanou metodu s daty, které získá z fake backendu. Nakonec je nutné data porovnat pomocí třídy Asert a metody AreEqual. Tato třída je součástí Unit test frameworku. Ukázku tohoto přístupu uvádím níže.

```
public void GetCustomerDeliverySites()
{

    Facade target = new Facade();
    CustomerTO customer = Serialization.DeserializeFromXml((string)TestContext.DataRow["
        Customer"]);
    string expected = (string)TestContext.DataRow["expectedResult"];
    string actual = Serialization .SerializeAnObject(target.GetCustomerDeliverySites(customer));

    Assert.AreEqual(expected, actual);
}
```

Výpis 3: Příklad použití Microsoft Frameworku pro Unit testy

Pokud je vše v pořádku, test prochází. V případě špatného zásahu do kódu testované metody test neprojde.

5.1.8 Framework pro nahrávání dat

Našim hlavním úkolem bylo vytvořit framework pro automatické nahrávání scénářů a automatické nahrávání dat do FBS. Jak jsem uvedl výše, pro každou testovanou metodu je třeba vytvořit scénářovou tabulku, což vzhledem k množství metod může při ručním vytváření zabrat mnoho času. Stejně tak toto platí i pro data nahrávána do FBS. Náš tým má k dispozici testovací webové prostředí, ve kterém je možné testovat výslednou aplikaci. Obsahuje jednoduché webové rozhraní, kde si programátor vybere, co chce testovat. Do tohoto rozhraní jsme přidali pár komponent, kde si programátor může vybrat, zda data nahrávat do FBS nebo zda nahrávat scénáře pro Unit testy. Všechno nahrávání probíhá za běhu aplikace tzn. programátor spustí část aplikace, ze které chce získat data, a ta jsou následně nahrávána do FBS nebo do scénářových tabulek. Ve výsledku tedy programátor používá program jako běžný uživatel, ale všechny jeho akce jsou zaznamenány. Pro vytváření scénářových tabulek bylo třeba vytvořit i testovací fasádu, kde do každé metody bylo třeba přidat volání frameworkové metody SaveToTable, která má za úkol vytvořit databázovou tabulku, která nese název dané metody a do této tabulky zapsat vstupy a výstupy metody. Pokud tabulka již existuje, jsou porovnána data v tabulce a na tomto základně jsou data buď zapsána nebo, pokud se data shodují, pokračuje se další metodou.

Tímto způsobem je možné získat data pro testování prakticky okamžitě. Testovací fasáda je volána jen v případě nahrávání scénářů. Níže uvádím ukázku použití frameworku.

```
public CustomerTO GetCustomer(string code)
{
    UnitTestsFramework.UnitTestDataStore save = new UnitTestsFramework.UnitTestDataStore();
    var  result  = CustomerBE.GetCustomer(code);
    save.SaveToTable(code, result);

    return result ;
}
```

Výpis 4: Ukázka využití našeho frameworku na fasádní vrstvě

5.2 Porovnání grafových knihoven jqPlot a HighCharts

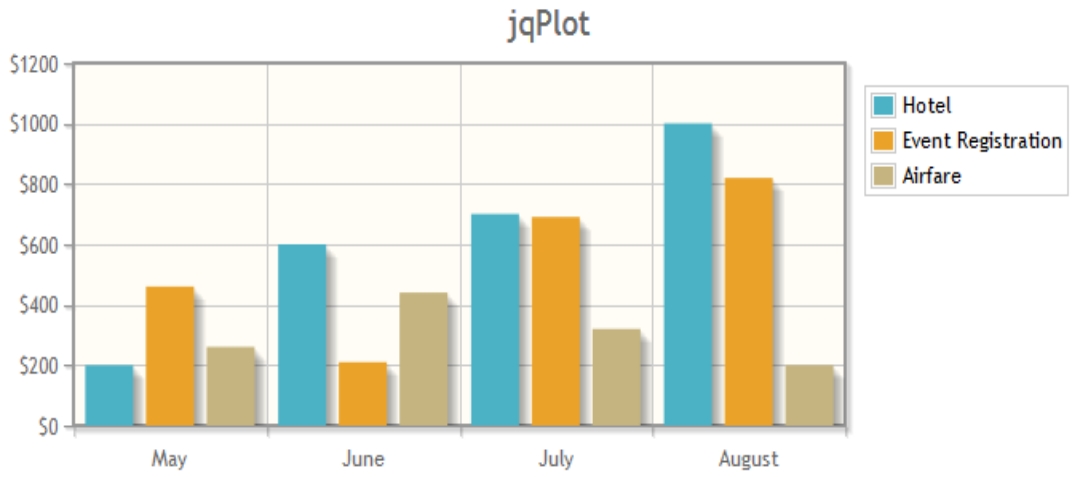
Dalším úkolem, který jsem vykonával sám, bylo porovnání grafových knihoven. V naší aplikaci má zákazník možnost zobrazit si různá data graficky např. spotřebu elektrické energie za nějaký časový interval. K těmto účelům tým Online využil skriptování na klientovi a zvolil javaScriptovou knihovnu jqPlot, která je poskytována zdarma pro jakékoliv využití. Mým úkolem bylo porovnat tuto knihovnu s javaScriptovou knihovnou HighCharts, která ovšem není pro komerční využití zdarma. Právě z těchto ekonomických důvodů jsem měl zjistit co nejvíce rozdílů, výhod a nevýhod a nakonec provést implementaci grafů v naší aplikaci za pomoci knihovny HighCharts.

5.2.1 Grafová knihovna jqPlot

Jde o sofistikovanou knihovnu, založenou na js frameworku jQuery. Nabízí mnoho typu grafů jako např. sloupcový, koláčový, spojnicový a mnoho dalších. Níže uvádím pár základních vlastnosti této knihovny.

- vícenásobné osy
- speciální typy grafů
- formátování popisků os
- Tooltips - zobrazení okamžité hodnoty při přidržení kurzoru na části grafu
- možnost rozšíření pomocí pluginů
- podpora mnoha prohlížečů - IE 7+, Firefox, Safari, Opera
- vykreslování grafů pomocí Canvasu, možnost animací grafu

jqPlot nabízí ještě mnoho dalších vlastností jako např. možnost stylování grafu. Důležitou vlastností této knihovny je způsob vykreslování, což obstarává technologie Canvas. Díky Canvasu můžeme s pomocí javaScriptu vykreslovat jednoduché tvary přímo do klientského prohlížeče. Podstatnou nevýhodou však je nemožnost přistupovat k nakreslenému tvaru. Pokud kupříkladu uvažujeme jednoduchý čtverec, který nakreslíme pomocí Canvasu v červené barvě a chceme jej později přebarvit na modro, je nutné nejdříve smazat celou kreslící plochu (plátno) a nakreslit čtverec znova v nové barvě, tedy modré. Při každé akci uživatele tedy dochází k překreslení celého grafu, což je zbytečné plýtvání prostředky. Tato vlastnost grafové knihovny jqPlot značně omezuje dynamiku kódu. Proto bych vykreslování Canvasem označil spíše za nevýhodu. V naší aplikaci jsou tedy všechny změny v grafu řešeny následným překreslením. Pokud si kupříkladu nechá uživatel vykreslit graf spotřeby elektrické energie a v dalším kroku chce k tomuto grafu zobrazit i venkovní teploty, je nutné načíst data pomocí Ajaxu a následně graf nakreslit znovu.



Obrázek 1: Ukázka jqPlot grafu

5.2.2 Implementace

jqPlot nabízí více možností implementace. Základní struktura grafového objektu je reprezentována strukturou JSON, kde jednotlivé atributy objektu představují vlastnosti grafu. Data, která graf zobrazuje, musí být reprezentována jako pole polí, kde každé dílčí pole představuje jednu sérii grafu. Tato data musí být grafu předána při inicializaci objektu a to z důvodu vykreslování grafu Canvasem. Níže uvádím jednoduchou ukázku vytvoření grafu.

```
$(document).ready(function()
{
    var plot1 = $.jqplot('HTMLdiv', [[3,7,9,1,4,6,8,2,5]]);
});
```

Výpis 5: Příklad inicializace jqPlot grafu

Tímto kódem vykreslíme jednoduchý spojnicový graf s jednou sérií, který bude vykreslen do elementu HTMLdiv webového dokumentu. Jak jsem výše zmiňoval, je možné grafu nastavit různé parametry, jako například typ grafu, popisy os atd. Vytvoříme tedy nový konfigurační objekt, který bude tyto parametry reprezentovat a přidáme jako parametr při inicializaci. U předchozího příkladu tento konfigurační objekt chybí, proto je graf vytvořen s přednastavenými parametry v rámci knihovny jqPlot. Níže uvádím příklad inicializace grafu s konfiguračním objektem.

```
$(document).ready(function()
{
      var config =
      {
          title : 'Graf s configem',
          series: [
                      {label:'serie1'}
                  ],
          axes: {
                      xaxis:
                      {
                          renderer: $.jqplot.CategoryAxisRenderer,
                          ticks: [1,2,3,4]
                      },
                      yaxis:
                      {
                          tickOptions: {formatString: '$\%d'}
                      }
                }
      }

    var plot1 = $.jqplot('HTMLdiv', [[3,7,9,1,4,6,8,2,5]], config);
});
```

Výpis 6: Příklad inicializace jqPlot grafu s konfiguračnímm objektem

Tento přístup má výhodu v tom, že si konfigurační objekty můžeme připravit předem pro každý graf, který se chystáme vykreslovat a za běhu aplikace jen vybrat, který konfigurační soubor se použije při vykreslování grafu. Tento přístup jsme také využili v naší aplikaci.

5.2.3 Grafová knihovna HighCharts

Jde o knihovnu napsanou v nativním js, s možností využívat různých js frameworků např. jQuery, MooTools atd. Podobně jako jqPlot nabízí velké množství typů grafů a také podporuje mnoho parametrů. Struktura vstupních dat a konfiguračních objektů je stejná jako u konkurenčního jqPlotu. Níže uvedu pár odlišných vlastností.

- podpora více prohlížečů: IE 6+, FireFox 2.0+, Chrome 1.0+, Safari 4.0+, Opera 9.0+, iOS(Safari) 3.0+, Android 2.0+
- vykreslování pomoci SVG, VML nebo Canvasu
- možnost přistupovat k parametrům grafu za běhu aplikace
- nativní podpora tisku a exportu grafu do různých formátů

HighCharts má tedy možnost dynamického přístupu ke svým parametrům za běhu programu díky SVG a VML vykreslování. SVG je technologie založená na vektorovém vykreslování, jehož struktura je založená na XML. Vykreslené objekty se stávají součástí DOM a jsou editovatelné pomocí js. Tento přístup jsem shledal jako velkou výhodu, jelikož můžeme i po vykreslení grafu manipulovat s jeho prvky a při jakékoliv změně není nutné graf smazat a nakreslit znovu. Další výhodou je přívětivější graf, a to z důvodu větších grafických možností technologie SVG. Jako příklad lze uvést využití gradientů, které s technologií Canvas nakreslit nelze. SVG podporuje většina moderních prohlížečů, nastává, ale problém se staršími prohlížeči od Microsoftu, které SVG nepodporují. Pro tyto prohlížeče HighCarts využívá technologii VML, což je obdoba SVG od Microsoftu. Protože HighCharts podporuje i platformu Android bylo nutné implementovat vykreslování pomocí Canvasu. HighCharts také podporuje stylování grafu, a to pomoci samostatného objektu. Mohu říci, že toto stylování je více sofistikované oproti konkurenčnímu jqPlot, a to zejména díky využití technologie SVG.

Obrázek 2: Ukázka HighCharts grafu

5.2.4 Implementace

Další částí mého úkolu byla implementace grafů za pomocí knihovny HighCharts do naší aplikace. Využil jsem tedy znalostí, které jsem popsal výše a vytvořil sadu konfiguračních objektů. Každý konfigurační objekt představoval jiný typ grafu. Podle uživatelské akce jsem rozlišil, jaký konfigurační objekt využít a nakonec jsem graf spolu se získanými daty vykreslil. Níže uvádím příklad inicializace grafu.

```
var config =
{
    chart:
    {
        renderTo: 'HTMLdiv';
    },
    series:
    [
       {
            name: 'Spojnicova serie',
            data: [200, 600, 700, 1000],
            type : 'spline'
       }
    ],
    xAxis :
    {
        categories: [jablka, hrusky]
    }
}

var plot = new Highcharts.Chart(config);
plot.addSeries({name: 'sloupcova serie', data: [150,2,40,100], type: 'column'});
```

Výpis 7: Příklad inicializace HighCharts grafu s konfiguračnímm objektem

V příkladu výše kromě vytvoření grafu můžeme na posledním řádku vidět i možnost dynamického přidání série po vytvoření grafu.

5.2.5 Zhodnocení

Po uvážení všech výhod a nevýhod obou knihoven bylo třeba rozhodnout, zda zůstat u jqPlotu či přejít na konkurenční HighCharts. Výhody knihovny HighCharts vycházejí hlavně z použití technologie SVG pro renderování. HighCharts je plně dynamický a nabízí pokročilé možností stylování grafu. Navíc podporuje větší škálu prohlížečů a nabízí kompletní API dokumentaci. Jeho hlavní nevýhodou je ale cena licence, která činí 360$ pro jednoho vývojáře či 80$ pro jeden projekt, a proto také byl přechod na HighCharts zamítnut. I přes své výhody by nepřinesl takové zjednodušení kódu či jiná vylepšení, která by usnadnili či vylepšili současnou aplikaci. Proto jsme se rozhodli, že by bylo zbytečné

kupovat licenci pro HighCharts. Tímto byl můj úkol splněn a všechny mé informace a implementace byly archivovány.

5.3 Převod prezentační vrstvy do MVC

V době psaní této bakalářské práce jsem na tomto úkolu pracoval, proto zde nebudu uvádět žádné zhodnocení či závěr, pouze popíšu celou ideu a můj postup při refactoringu. Základní impulsem pro převod prezentační vrstvy do MVC byla myšlenka o vylepšení struktury celého kódu. Jak jsem již uvedl, velká část prezentační vrstvy byla vytvořena v js (viz. 4) a kód této vrstvy začal zbytečně nabírat na složitosti a nepřehlednosti. Jako nejvhodnější řešení se nabízelo refactorovat tuto vrstvu do MVC a omezit tak skriptování na nezbytné minimum. V js by po přepsání mělo zůstat jen vykreslování grafu a jQuery efekty.

5.3.1 Návrhový vzor MVC

Jde o architektonický vzor, který popisuje aplikaci jako propojení tří komponent tak, že záměna či úprava kterékoliv z těchto komponent má minimální vliv na ostatní komponenty. Tento přístup je velmi obecný, ale velmi správný a efektivní, neomezuje se na implementaci v konkrétním jazyce ani na platformě. V principu jde o interakci uživatelského rozhraní (view), řídící logiky (controller) a datového modelu (model), odtud tedy název MVC.

- **Model** - jedná se o reprezentaci dat, se kterou aplikace pracuje. Většinou se jedná o doménové objekty či transfer objekty, které reprezentují objekty reálného světa jako např. Customer, Account atd. Model jako takový obecně neřeší přístup k databázi.
- **View** - Slouží pro prezentaci dat modelu uživateli aplikace. U webových aplikací se jedná o HTML stránky, které bývají dynamicky generovány.
- **Controller** - Tvoří řídící logiku aplikace. Reaguje na požadavky uživatele a zajišťuje zobrazení view s požadovanými daty modelu, se kterými může také manipulovat. U webových aplikací, bývají požadavky odesílány pomocí HTTP requestu, které controller odchytává a na základně požadavku vyžádá data od modelu a zobrazí view se získanými daty. U složitějších aplikací bývá také zařazen tzv. front controller, který odchytává požadavky uživatele a tyto přeposílá příslušným controllerům.

5.3.2 Obecný princip MVC

Obecně můžeme uvažovat vstup uživatele do jednoduché webové aplikace, kde první request pro zobrazení úvodní stránky odchytí controller, ten požádá o příslušná data model a zobrazí view (úvodní HTML stránku). Uvažujme, že na této stránce (view) se nachází přihlašovací formulář, který následně uživatel vyplní a odešle. Tímto vznikne další HTTP request, který má jako parametry uživatelské jméno a heslo, které byly zadány uživatelem. Tento request je odchycen controllerem. Controller nyní požádá model o ověření

přihlašovacích údajů. Model tyto údaje vyhodnotí oproti databázi a vrátí controlleru true, pokud jsou přihlašovací údaje platné nebo false, pokud jsou neplatné. Controller na tomto základě zobrazí uživateli view, které reprezentuje úspěšné nebo neúspěšné přihlášení do aplikace na základě předchozích předpokladů.

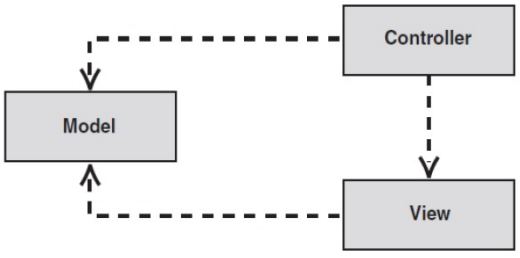
5.3.3 Postup při refactoringu

Pro jednodušší práci při vývoji aplikací v MVC bylo vytvořeno více frameworků. Tým Online využil hotové řešení Microsoftu, tedy ASP.NET MVC. Mým úkolem tedy bylo vytvořit modely, controlery a view. Před přebráním úkolu již bylo část kódu převedeno do MVC, proto jsem nezačínal úplně od píky. Pro omezení velké části js jsem přesunul klientské kódy na server a přepsal do C#. Klientské efekty a další komponenty citlivé na odezvu musely zůstat v js. Pro vytváření view byl vhodný Microsoft Razor, což je takové rozšíření původního HTML o C# či Visual Basic kód. Výsledkem je dynamický kód. Níže uvádím ukázku Razoru.

```
@{
  var name = "MVC";
  <div>
      This project is made by @nazev
  </div>
  }
```

Výpis 8: Ukázka práce s MS Razor

Je třeba s tímto ale nakládat opatrně a s rozvahou, protože může vzniknout stejně neudržitelný kód jako tomu je v js. Klíčem k úspěchu v naší aplikaci je minimalizovat C# kód v Razoru a využít co nejvíce partialView. Mé první kroky tedy vedly k úpravám některých view a to tak, že jsem převedl stávající webForms do Razoru. Další věcí, která stojí čistě na js je načítání dat pro graf na klientovi. Docházelo zde k načtení dat ze serveru, ale příprava dat pro graf již probíhala na klientovi. Proto bylo vhodnější převést tento kód na server, který připraví data, serializuje a klient po té jen načte připravená data. Pro všechna



Obrázek 3: Schéma architektury MVC

partial view bylo nutné vytvořit příslušné controllery a modely. Datová vrstva (viz. 4.1) již obsahovala všechny potřebné transfer objekty, které definují všechny entity reálného světa. Tyto jsem také využil pro modely. Každý model tak obsahoval sadu proměnných, které byly typu příslušného transfer objektu. Controllery byly navrženy tak, že obsahovali referenci na fasádní vrstvu, díky které bylo možné komunikovat s business vrstvou aplikace. Funkce controlleru tedy spočívala v tom, že při jeho vyvolání byla zavolána metoda fasádní vrstvy, její výstup uložen do příslušného modelu a vraceno view, které reprezentovalo data tohoto modelu.

5.3.4 Refactoring Grafu

Jedním z úkolů při refactoringu do MVC byl také převod konfiguračních objektu a přípravy dat pro graf na server. Jak jsem již uvedl dříve, příprava dat pro graf i inicializace konfiguračních objektů probíhala na klientovi v js. Pro větší efektivitu jsem převedl celou strukturu konfiguračního objektu na server. To obnášelo vytvoření třídy GraphTO (Modelu) v jazyce C#, která obsahovala všechny parametry původních js konfiguračních objektů. Dále bylo nutné vytvořit pomocnou metodu getGraphConfig, která podle vstupních parametrů vytvořila objekt třídy GraphTO. Na straně klienta, byly shormážděny všechny potřebné parametry díky interakci s uživatelem a jejich odeslání na server zabezpečoval AJAX call. Po odeslání byl zavolán příslušný controller, který zpracoval předaná data. V controlleru docházelo k získání dat pro graf a získaní konfigurací díky pomocné metodě getGraphConfig. Tato data a konfigurace byla zabalena do dalšího anonymního objektu, který byl po té serializován do datového typu JSON a vrácen zpět na klienta, kde byl původně volán AJAX call. Na klientovi po té byl na základě vracených dat vykrelen graf pomocí js. Efektivita celé této práce tedy spočívala v minimalizaci js a převodu zpracování dat na server.

6 Závěr

6.1 Uplatněné znalosti získané během mého studia

Jelikož jsem byl ve firmě Tieto zařazen na pozici Software developer, využil jsem velkou škálu svých vědomostí nabytých studiem na vysoké škole. Jednalo se především o znalosti z oblasti programování a vývoje serverových a klientských aplikací. Především jsem programoval na platformě .NET a proto si myslím, že velmi užitečnými a přínosnými pro mě byly předměty Programovací Jazyky II a Architektura technologie .NET. Jak jsem zmínil, podílel jsem se i na programování klientských aplikací a to v JavaScriptu, proto pokládám také Vývoj internetových aplikací za přínosný předmět. Při práci s FBS jsem využíval znalosti z okruhů databázové problematiky, o které jsem také dostal mnoho informací na vysoké škole. V této práci se také zmiňuji o návrhových vzorech Fasáda a MVC, se kterými jsem se setkal v předmětu Vývoj informačních systémů.

6.2 Znalosti, které jsem během praxe postrádal

Po dobu mé praxe jsem poprvé přišel do styku s práci v týmu, a také práci se sdíleným zdrojovým kódem. Chyběla mi znalost knihoven jqPlot a HighCharts, které jsem měl za úkol porovnat, a proto jsem je musel nastudovat z webové dokumentace. Také jsem měl pouze základní povědomí o Unit testování, a proto jsem si jej musel rozšířit. Z pohledu využitých programovacích jazyků a technologií si myslím, že jsem netrpěl žádnými většími nedostatky.

6.3 Zhodnocení dosažených výsledků a odborné praxe

Absolvování odborné praxe ve firmě Tieto považuji za velmi pozitivní a přínosnou zkušenost. Vyzkoušel jsem si práci na rozsáhlém projektu v týmovém prostředí a sdíleném zdrojovém kódu, s čímž jsem předtím neměl zkušenosti. Díky práci na tomto projektu jsem si utvrdil mé zkušenosti a také vědomosti z oblasti tvorby informačních systémů. Seznámil jsem se také s metodikou Scrum, která byla v této společnosti zavedena. Získal jsem zkušenosti z prezentací mých dokončených úkolů, a také přehled nad průběhy meetingů a komunikací týmu.

7 Reference

[1] Tieto Czech s.r.o, *Tieto - O nás*, [Online], [citace:29.března2013], Dostupné z: `http://www.tieto.cz/tieto-o-nas`.

[2] Chris Leonello, *jqPlot - pure javaScript plotting*,[Online],[citace: 29. března 2013],Dostupné z: `http://www.jqplot.com/index.php`.

[3] Highsoft Solutions AS, *HighCharts - products*, [Online], [citace: 29. března 2013], Dostupné z: `http://www.highcharts.com/products/highcharts`.

[4] Microsoft, *Model-View-Controller*, [Online], [citace: 29. března 2013], Dostupné z: `http://msdn.microsoft.com/en-us/library/ff649643.aspx`.

[5] Osherove, Roy, *The art of unit testing* , Manning Pub. Co., 2009, ISBN: 1933988274.